Videojuegos

Curso de Diseño y Programación

Nº 11 5,99 euros



***Implementación* del protagonista**

Manejo de *niveles BSP*

Las distintas herramientas de *FruityLoops*



11







Curso práctico de Videojuegos

AUTOR DE LA OBRA
Marcos Medina
DIRECCIÓN EDITORIAL
Eduardo Toribio
etoribio@iberprensa.com
COORDINACIÓN EDITORIAL
Eva-Margarita García
eva@iberprensa.com
DISEÑO Y MAQUETACIÓN
Antonio Gª Tomé
PRODUCCIÓN
Marisa Cogorro

SUSCRIPCIONES
Tel: 91 628 02 03
Fax: 91 628 09 35
suscripciones@iberprensa.com

FILMACIÓN: Fotpreim Duvial
IMPRESIÓN: Gráficas Don Bosco
DUPLICACIÓN CD.ROM: M.P.O.
DISTRIBUCIÓN
S.G.E.L.
Avda. Valdelaparra 29 (Pol. Ind.)
28108 Alcobendas (Madrid)
Tel.: 91 657 69 00

EDITA: Iberprensa
www.iberprensa.com

CONSEJERO
Carlos Peropadre

REDACCIÓN, PUBLICIDAD Y ADMINISTRACIÓN
C/ del Río Ter, 7 (Pol. Ind. "El Nogal")
28110 Algete (Madrid)
Tel.: 91 628 02 03
Fax: 91 628 09 35
(Añada 34 si llama desde fuera de España.)

DEPÓSITO LEGAL: M-35934-2002
ISBN: Coleccionable: 84 932417 2 5
Tomo 2: 84 932417 4 1
Obra Completa: 84 932417 5 X

NOTA IMPORTANTE:
Algunos programas incluidos en los CD de "Programación y Diseño de Videojuegos" son versiones completas, pero en otros casos se trata de versiones demo o trial, versiones de evaluación que Iberprensa quiere ofrecer a nuestros lectores. No se trata en ningún caso de las versiones comerciales de los programas, y las hemos incluido para dar al lector la oportunidad de conocer y probar esos programas y que así pueda decidir posteriormente si desea o no adquirir las versiones comerciales de cada uno.

Aprende divirtiéndote

Bienvenidos a **Programación y Diseño de Videojuegos**, la primera obra coleccionable cuyo objetivo es formar al alumno en las principales técnicas relacionadas en el desarrollo completo de un videojuego.

A lo largo de la obra el lector aprenderá programación a nivel general y a nivel específico con ciertas herramientas y lenguajes, aprenderá a trabajar con aplicaciones de retoque de imagen y también de diseño 3D y animación. Descubrirá las aplicaciones profesionales más importantes de audio y conocerá la historia de lo que se denomina "la industria del videojuego", los últimos 20 años, los juegos que marcaron un avance, sus creadores y en general la evolución del videojuego.

Pero además, esta obra tiene un segundo objetivo, desarrollar y potenciar la creatividad del lector, nosotros a lo largo de las diferentes entregas pondremos las bases y tú pondrás tu ingenio, tu creatividad y tu capacidad de mejorar.

Comienza aquí un viaje de 20 semanas articulado en 400 páginas y 20 CD-ROMs cuya finalidad es proporcionar las bases mínimas para después cada uno continuar su camino.

Recuerda que para alcanzar el éxito necesistas cumplir tres condiciones: que te gusten los juegos, poseer cierta dosis de creatividad y finalmente capacidad de estudio.

Una la cumples seguro.

sumario

201 Zona de desarrollo
Empezamos a aprender a implementar al protagonista del juego, controlando su movimiento y el de la cámara.

205 Zona de gráficos
Continuamos fabricando los distintos elementos del juego; en esta ocasión, creamos los restantes seres que habitan los terrenos de combate.

209 Zona de audio
Seguimos creando loops y viendo distintas herramientas que hacen de FruityLoops un potentísimo programa.

211 Blitz 3D
Después de realizar nuestros modelos, vamos a texturizarlos y a pintarlos con la ayuda de los *brushes*.

215 Tutorial
Empezamos una nueva serie de tutoriales relacionados con el desarrollo viendo el manejo de niveles BSP.

217 Historia del videojuego
Seguimos con los juegos de estrategia, conociendo la historia de los RTS o juegos de estrategia en tiempo real.

219 Cuestionario
Cada semana un pequeño test de autoevaluación, en el próximo número encontrarás las respuestas.

220 **Contenido CD-ROM**
Páginas dedicadas a la instalación y descripción del software que se adjunta con cada coleccionable.

11

PARA ENCUADERNAR LA OBRA:

- *Para encuadernar los dos volúmenes que componen la obra "Programación y Diseño de Videojuegos" se pondrán a la venta las tapas 1 y 2.*
- ***Tapas del volumen 2 ya a la venta.***
- ***Los suscriptores recibirán las tapas en su domicilio sin cargo alguno como obsequio de Iberprensa.***

SERVICIO TÉCNICO:
Para consultas, dudas técnicas y reclamaciones Iberprensa ofrece la siguiente dirección de correo electrónico: games@iberprensa.com

PETICIÓN DE NÚMEROS ATRASADOS:
El envío de números sueltos o atrasados se realizará contra reembolso del precio de venta al público más el coste de los gastos de envío. Pueden ser solicitados en el teléfono de atención al cliente 91 628 02 03

Controlando al protagonista (I)

En esta primera entrega acerca de cómo implementar a nuestro protagonista en el juego, empezaremos por controlar su movimiento y el de la cámara, tratando también la actualización de su estado durante la partida.

(■) MOVIENDO A NUESTRO PROTAGONISTA

Antes de empezar, vamos a situar todo el procedimiento de control del jugador en la función "control_jugador_principal()" en el módulo "jugador_principal.bb".

La bionave de combate es movida por el jugador mediante el ratón y/o de las teclas del cursor. Debemos, sin embargo, estudiar previamente cómo se ha diseñado este movimiento para saber qué instrucciones debemos utilizar. Sabemos que la premisa más importante es que la bionave siempre está en movimiento. Por lo tanto, lo más sensato será utilizar el comando "MoveEntity" para desplazarla. Otra cuestión es cómo la movemos. Básicamente, todo el control de la bionave se centra en el uso del ratón a través de su desplazamiento y la pulsación de los botones, aunque podemos añadir el control por teclado fácilmente. Pero centrémonos en el uso del ratón. A pesar de que la obtención de datos a través de periféricos externos la estudiaremos más detenidamente en la sección "Blitz3D" de una próxima entrega, adelantaremos su uso en este número de una forma sencilla y fácil de comprender.

(■) ROTANDO LA BIONAVE CON EL RATÓN

Lo que primero nos interesa es cómo obtener el desplazamiento del ratón y convertirlo en datos numéricos, los cuales servirán para modificar las coordenadas de la bionave. Solamente nos interesa el movimiento de los ejes horizontales del ratón para modificar la rotación de la nave en su eje Y. Así que, desplazándolo de izquierda a derecha, hacemos que la bionave gire para cambiar de rumbo ya que, si recordamos, siempre está en movimiento.

Es importante resaltar que, para crear un punto de rotación central en la nave, es imprescindible crear un pivote. El procedimiento para su creación lo implementamos en el momento de cargar el fichero y crear el modelo en la función "crear_modelos()" del módulo "funcpantaudio.bb":

```
Function crear_modelos()
  bionave=LoadMesh ("c:\zone of
  fighters\modelos\bionave.3ds")
  pivote_bionave=
  CreatePivot(bionave)
…
```

De vuelta al módulo de control, comenzamos la función almacenando en la variable flotante YY# el desplazamiento horizontal absoluto del ratón mediante la función "MouseXSpeed", la cual detecta los cambios en su movimiento. El valor obtenido por esta función es un número entero; debemos convertirlo en un número flotante y dividirlo entre 7 para obtener valores decimales y así tener incrementos o decrementos de desplazamientos razonables.

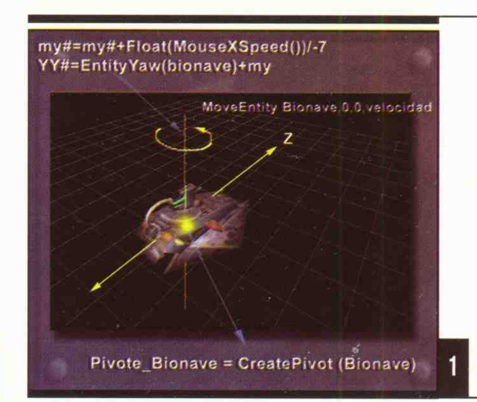
```
my#=my#+Float(MouseXSpeed())/-7
YY#=EntityYaw(bionave)+my
```

A continuación, ejecutamos la rotación de la bionave y su sombra en su eje Y. Además, hemos tenido en cuenta un incremento extra en la velocidad de rotación cuando la cámara se encuentra en el modo subjetivo o primera persona, controlado por la variable global "tipo_camara":

```
RotateEntity bionave,0,yy+
MouseXSpeed() , 0
```

De esta forma, todo el proceso queda como sigue:

```
If tipo_camara=1
    my#=my#+Float(MouseXSpeed())/-7
    YY#=EntityYaw(bionave)+my
    RotateEntity bionave,0,yy+
    MouseXSpeed(),0
    RotateEntity sombra_bionave,
    0,yy,0
  Else
    my#=my#+Float(MouseXSpeed())/
    -7 : my=my/2 ;➡ Mitad de
    velocidad de giro para 3ª
    persona
    YY#=EntityYaw(bionave)+my
    RotateEntity bionave,0,yy,0
    RotateEntity sombra_bionave,
    0,yy,0
EndIf
MoveMouse GraphicsWidth()
Shr 1,GraphicsHeight() Shr 1
```



Representación gráfica de la rutina de rotación y desplazamiento de la bionave.

La función "MoveMouse" colocará el cursor (invisible) siempre en el centro de la pantalla, así nunca perderemos el desplazamiento del ratón (Fig. 1).

(■) DESPLAZAMIENTO Y ACELERACIÓN

En el diseño del juego contemplamos la idea de que la bionave del protagonista se encuentra en continuo desplazamiento. El único control que el jugador puede tener sobre este desplazamiento es aumentar o disminuir la aceleración. Para ello, elegimos de nuevo el ratón como vehículo principal de control. Utilizaremos entonces el botón derecho para aumentar la aceleración. Si lo pulsamos aumentará y si lo soltamos disminuirá hasta una velocidad inicial establecida. También debemos contemplar la posibilidad del uso del teclado por parte del jugador, ya que siempre es necesario implementar en el juego todas las opciones posibles de control. Así que utilizaremos la tecla del cursor "arriba" para aumentar también la aceleración.

Las variables que utilizaremos para dotar a la bionave de movimiento son: *velocidad#* y *aceleración#*. Ambas son inicializadas en "Definiciones.bb" con los valores 0.8 y 0.2 respectivamente.

Para implementar este control utilizamos una sentencia condicional:

```
If (MouseDown(boton2) Or
KeyDown(200)) Then
      velocidad#=velocidad+aceleracion#
      MoveEntity bionave,0,0,velocidad
   Else
      velocidad=velocidad-(aceleracion#*2)
      MoveEntity bionave,0,0,velocidad
EndIf
```

Si pulsamos el botón derecho del ratón o pulsamos el cursor "arriba" incrementamos la velocidad según el valor contenido en *aceleración#* (0.2). Si no, reducimos la velocidad el doble de rápido que aceleramos:

```
velocidad=velocidad-(aceleracion#*2)
```

De esta forma, evitamos una desaceleración brusca y conseguimos un movimiento suave. Pulsemos el botón o no, siempre desplazamos la bionave en su eje Z; es decir, hacia delante o hacia atrás, ya que el resto de direcciones lo determinará el giro:

```
MoveEntity bionave,0,0,velocidad
```

Es necesario añadir una condición más a la anterior sentencia. Se trata de controlar la velocidad máxima para que al acelerar haya un tope. Utilizaremos otra variable llamada *velocidad_max%* la cual inicializamos con un valor de 15:

```
If (MouseDown(boton2) Or
KeyDown(200)) And velocidad <
velocidad_max Then
```

De esta manera conseguimos que la velocidad actual de la bionave no supere el valor 15.

A su vez, hemos de controlar también la velocidad mínima, por ejemplo 4; de esta manera garantizamos un desplazamiento mínimo continuo. Debemos añadir, seguidamente, una sentencia más:

```
If velocidad<4 velocidad=4
```

Antes de controlar el factor de gravedad, debemos actualizar las coordenadas de la bionave y la posición de la cámara llamando a la función "mueve_camara":

```
mueve_camara (bionave,
velocidad_seguimiento)
```

Para el correcto funcionamiento de la función es imprescindible pasarle como parámetros la entidad a la cual seguirá, en este caso la bionave, y a la velocidad que lo hará ("velocidad_seguimiento") (Fig. 2).

(■) CONTROLANDO LA CÁMARA. FUNCIÓN "MUEVE_CAMARA"

El juego soporta cuatro vistas de cámara diferentes: subjetiva, tercera persona, cenital y seguimiento en tercera persona. En todos los casos existe una particularidad común y es que la cámara sigue en todo momento a la bionave. Para realizar este procedimiento utilizaremos la instrucción "PointEntity", la cual implementaremos en la función "mueve_camara" situada en el módulo "Fjuego.bb".



La distancia entre la cámara y la bionave en 3ª persona depende de la variable "Velocidad_seguimiento".

Los parámetros de entrada de esta función son: el objetivo a seguir ("objetivo") y la velocidad de seguimiento de la cámara ("velocidad_camara"):

```
Function mueve_camara
(objetivo, velocidad_camara)
```

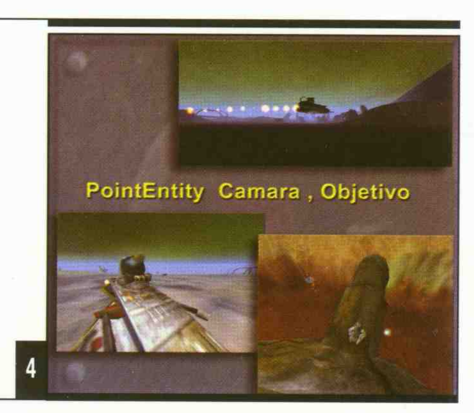
Posteriormente actualizamos la posición de la bionave almacenándola en las variables *X#* , *Y#* y *Z#*:

```
x#=EntityX(bionave)
y#=EntityY(bionave)
z#=EntityZ(bionave)
```

Continuamos, calculando las coordenadas de la cámara con



Fórmulas empleadas para calcular las coordenadas de la cámara con respecto al objetivo (bionave). Además, permiten desplazar y rotar la cámara de una forma suave.



Independientemente de la vista escogida, la cámara siempre apuntará al objetivo (bionave) mediante la instrucción "PointEntity".

respecto al objetivo a seguir. Para ello, determinaremos un desplazamiento según la posición actual de la cámara y del objetivo y lo almacenaremos en las variables *CX#* y *CZ#*:

```
cx#=EntityX(camara)
cz#=EntityZ(camara)
objetivo_x#=EntityX(objetivo)
objetivo_z#=EntityZ(objetivo)
```

objetivo_x# y *objetivo_z#* almacenarán la posición del objetivo a seguir, en este caso la bionave, que servirán luego para actualizar la posición de la cámara de una forma suave y siempre dependiendo del contenido de la variable *velocidad_camara*. Para implementar estos desplazamientos utilizaremos las fórmulas que se muestran en la figura 3.

```
cx#=cx#+((objetivo_x#-cx#)
/velocidad_camara)
cz#=cz#+((objetivo_z#-cz#)
/velocidad_camara)
```

La altura de la cámara dependerá del tipo de vista que se haya seleccionado. De tal modo que, si la vista es subjetiva -en primera persona-, debemos situarla al nivel de la cabeza del protagonista. Para ello, situaremos la cámara 4 puntos más arriba del centro vertical de la bionave; es decir, "Y# + 4". Para el resto de las vistas, precisaremos en todo momento de la altura del terreno. Además, es necesario añadirle una altura extra para evitar una visión a ras de suelo y de esta manera situar la cámara por encima de la bionave. Debemos entonces utilizar una variable para almacenar esta altura adicional: *distanciaY_camara%*.

Este valor será el que determinará los tipos de vista cenital, tercera persona o seguimiento. Por defecto asignamos un valor de 158 en el módulo de "Definiciones.bb". Almacenaremos este valor en la variable *CY#* que será la que utilicemos finalmente:

```
cy#=distanciaY_camara
If tipo_camara=1 ; Primera persona
  PositionEntity camara,cx,y+4,cz
Else ; Otras vistas
  PositionEntity Camara,cx,
  TerrainY(terreno1,EntityX
  (bionave),EntityY(bionave),
  EntityZ(bionave))+cy,cz
EndIf
```

Finalmente, nos queda apuntar la cámara hacia el objetivo y actualizar la variable *y_bionave#*, la cual nos ayudará a situar la sombra de la bionave, ya que almacena la posición de ésta sobre el terreno:

```
PointEntity camara,objetivo
y_bionave=TerrainY(terreno1,
x,y+20,z)
```

(■) CAMBIANDO LAS VISTAS DE CÁMARA. FUNCIÓN "CAMBIAR_CAMARA"

Dentro del módulo "Fjuego.bb", hemos situado, a continuación de la anterior función, un procedimiento para actualizar la situación de la cámara en las distintas vistas disponibles para el usuario. La función adquiere el parámetro del tipo de cámara elegido almacenado en la variable global *tipo_camara*:

```
Function cambiar_camara (tipo_camara)
```

Después, simplemente utilizamos una sentencia "Select .. Case" para modificar los parámetros de la situación de la cámara según el tipo elegido.

Hay otros factores a tener en cuenta a la hora de cambiar las vistas y son controlar la visualización de la bionave, su sombra y cambiar el zoom de la cámara. Por lo tanto, en la vista subjetiva debemos ocultar la entidad bionave y su sombra para poder dar la sensación deseada. Para ocultar la bionave cambiaremos su opacidad a 0 con "EntityAlpha bionave, 0 " y para hacer lo mismo con la sombra utilizaremos " HideEntity sombra_bionave ". Es necesario controlar también si el jugador tiene activado o no el camuflaje, examinando la variable *camuflaje_bionave*:

```
Select tipo_camara
  Case 1
    distanciaZ_camara=-2
    velocidad_seguimiento=6
    EntityAlpha bionave,0
    HideEntity sombra_bionave
    CameraZoom camara,.54
  Case 0
    distanciaY_camara=107
    distanciaZ_camara=0
    velocidad_seguimiento=15
    If camuflaje_bionave=0
      EntityAlpha bionave,1
    Else
      EntityAlpha bionave,.1
    EndIf
      ShowEntity sombra_bionave
      CameraZoom camara,.8
  Case 2
    distanciaY_camara=700
    distanciaZ_camara=0
    velocidad_seguimiento=63
    If camuflaje_bionave=0
      EntityAlpha bionave,1
    Else
      EntityAlpha bionave,.1
```



Dos ejemplos de la vista en tercera persona.



6

Algunos ejemplos de vista de seguimiento en tercera persona.

```
   EndIf
     ShowEntity sombra_bionave
     CameraZoom camara,.8
   Case 3
     distanciaY_camara=62
     distanciaZ_camara=-139
     velocidad_seguimiento=12
   If camuflaje_bionave=0
     EntityAlpha bionave,1
   Else
     EntityAlpha bionave,.1
   EndIf
     ShowEntity sombra_bionave
     CameraZoom camara,.54
End Select
```

Para terminar la función es preciso devolver el valor de la variable *tipo_camara*:

```
Return (tipo_camara)
```

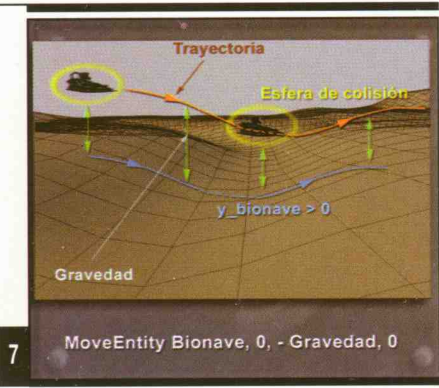
(■) CONTROLANDO EL FACTOR GRAVEDAD

Para determinar un entorno creíble es importante proporcionar ciertos elementos físicos, como por ejemplo la gravedad. Implementarla es realmente fácil. Sólo tenemos que mover a la bionave continuamente hacia el suelo decrementando su coordenada Y con el valor de "gravedad", pero sólo si está sobre el suelo; es decir, si supera el valor "Y=0":

```
If y_bionave>0
  MoveEntity bionave,0,-gravedad,0
EndIf
```

De todas formas, el sistema de colisiones tiene prioridad frente al sistema de coordenadas. Por lo tanto, aunque siguiéramos decrementando la coordenada Y, la bionave se pararía al tocar con el terreno. La sentencia condicional anterior sólo nos sirve para evitar, precisamente, que la coordenada "y_bionave" baje del valor 0 (Fig. 7).

Explicaremos la implementación de las colisiones con el entorno en una próxima entrega.



7

Esquema gráfico del funcionamiento del control de la gravedad.

(■) AÑADIENDO DETALLES. SOMBRA Y COMBUSTIÓN

Para terminar, podemos añadir otros detalles, como la sombra de la bionave o la combustión del motor.

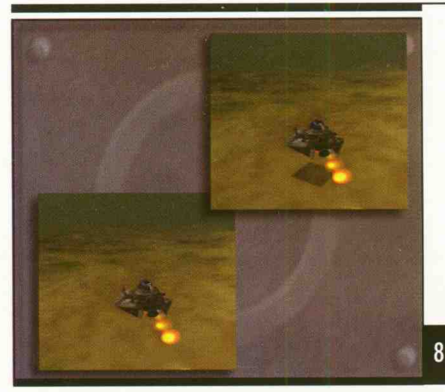
El hecho de añadir una sombra a la nave no se debe sólo a cuestiones estéticas, ya que la bionave se desplaza sobre el terreno en suspensión. Por lo tanto, es fundamental implementarla para proporcionar información adicional al jugador de la altura de la bionave. Utilizando, pues, la variable *y_bionave*, obtenemos la altura actual de ésta sobre el terreno. De este modo, sólo tenemos que colocar la sombra según los ejes X, Y y Z de la bionave y 15 puntos más abajo con referencia al eje Y (Fig. 8).

Si "y_bionave" vale "TerrainY(terreno1,x,y+20,z)", debemos posicionar la sombra en "y_bionave+5":



```
PositionEntity sombra_bionave,
x,y_bionave+5,z
```

Otro factor a tener en cuenta es proporcionar el efecto de la combustión del motor. Este hecho reforzaría aún más la idea de que la bionave está continuamente en movimiento. La mejor manera de implementar la combustión es a través de un emisor de partículas situada en el tubo de escape. Pero de este elemento hablaremos con mucho más detalle en una próxima entrega (Fig. 9).



8

A diferencia de la figura inferior, en la superior se puede apreciar la distancia de la bionave al suelo gracias al uso de sombras.

Hay muchos más aspectos que pueden ser implementados en las rutinas de movimiento y cámara del protagonista como pueden ser la aplicación de algún tipo de animación o colocar una nueva cámara que muestre la parte trasera de la bionave.



9

Un ejemplo de emisión de partículas para simular la combustión del motor.

En el próximo número...

... continuaremos controlando a nuestra bionave, programando las acciones de disparo, camuflaje y activación del escudo.

Fabricando los elementos del juego (II)

En este número terminaremos de realizar el Luny. Continuaremos fabricando el resto de seres que habitan los terrenos de combate como los "Shaark" y las plantas carnívoras "Dreecks".

CREANDO EL MAPEADO UV Y TEXTURIZANDO EL "LUNY" CON DEEP PAINT 3D

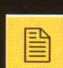
Para crear el mapeado UV realizaremos la misma operación que con el "Slunk" en LithUnWrap. Optimizaremos y aplicaremos un "UV mapping" "Planar" con la opción "Y top". Una vez salvado, lo cargamos con Deep Paint 3D para texturizarlo. Generamos un material nuevo con 256 x 256 de tamaño y aplicamos una imagen en blanco "Nothing" al canal de color ("C"). Ajustamos el modelo en la ventana y aplicamos un relleno con la textura *Skin 2* en *Texture Paints*. Para finalizar, pintaremos los detalles de la sangre y suciedad en la base de la planta utilizando la misma variación de pincel que en el modelo anterior; es decir, *Paste Oil* + en *Variations* con un pincel escalado al 15% (Fig.1).

Una vez pintado, exportamos la textura con el nombre "textura_luny.bmp" en el directorio "C.\juego_ZOF\luny\texturizado".

MODELADO DE UNA PLANTA SHAARK

Esta planta tiene un aspecto parecido al de las mazas utilizadas como armas en la Edad Media. No tiene movimiento alguno y aparecerá en el juego clavada en la tierra. Así que empezamos modelando el tronco que no es más que un cilindro de 2 "Stack" (Divisiones) y 8 "Slices". Para crear la base de la esfera de la planta, desplazamos los vértices que dividen las dos secciones del tronco hacia arriba. Seleccionamos los vértices superiores y, con la herramienta de escalado, los abrimos. A continuación, de los 8 vértices bajamos 4 de dos en dos y salteados como se muestra en la figura 2 para lograr una forma estrellada.

Para crear la cabeza de la planta elegimos una geoesfera de densidad 1 y la situamos sobre la base en forma de estrella que acabamos de modelar. Para formar los pinchos que irán clavados en la cabeza de la planta partimos de cilindros de una sola división y de 3 "Slices". Seleccionamos los vértices superiores y los unimos con "Ctrl" + "N" para formar la punta. Una vez modelada la espina la colocamos en cualquier lugar de la cabeza. Ya sólo nos queda duplicarla varias veces y repartir las nuevas espinas por la superficie de la esfera realizando operaciones de rotación y desplazamiento. Finalizamos, seleccio-

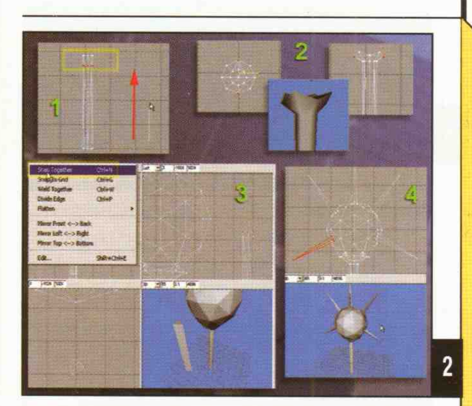


TRUCO

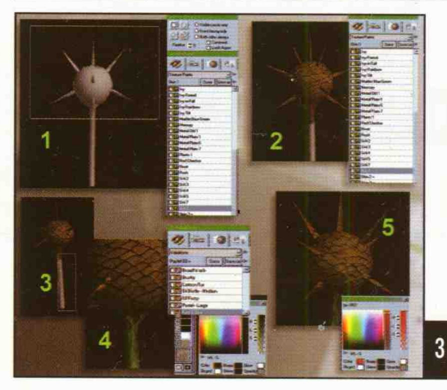
Podemos abrir una ventana con la perspectiva del modelo en LitUnWrap con la opción *Show Model* en el menú *Preview*.



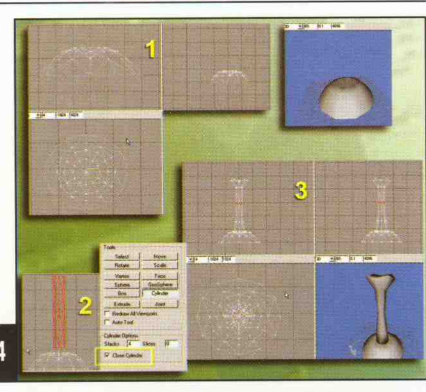
Después de cubrir con una textura el modelo es interesante la aplicación de detalle por medio del pincel.



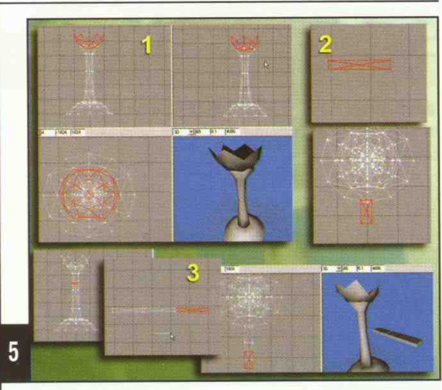
Procedimientos para el modelado de una planta Shaark.



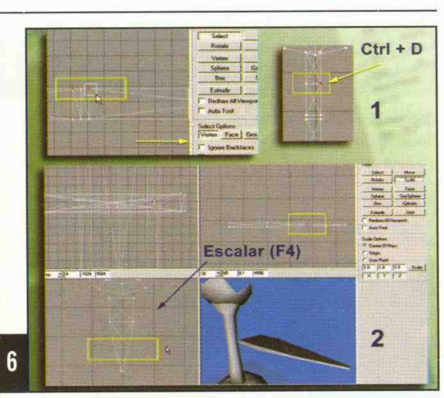
Procedimientos para el texturizado de un Shaark.



Los anillos de un cilindro nos servirán para dar forma al tallo de la planta.



La mitad de una geoesfera es ideal para obtener la base de la boca.



Para obtener la forma de las hojas a partir de tres cubos unidos podemos aplicar las herramientas de escalado y rotación en las uniones.

nando todas las piezas y agrupándolas para exportar como un sólo *mesh* y en formato .ms3D con el nombre "shaark.ms3d" en el directorio "C:\juego_ZOF\shaark\modelado\"

CREANDO EL MAPEADO UV TEXTURIZANDO CON DEEP PAINT 3D

El mapeado UV lo crearemos de nuevo con LithUnWrap, pero en esta ocasión, y una vez optimizado, aplicaremos un "UV Mapping" "Planar" en su eje X, para conseguir una plantilla que muestre una disección del modelo.

Exportamos en formato .OBJ y posteriormente lo cargamos en Deep Paint 3D para texturizarlo.

Al igual que en el modelo anterior, generamos una nueva textura de dimensiones 256 x 256 y aplicamos "Nothing" en el canal "C". Vamos a pintar con aspecto diferente el tronco y la cabeza de la planta. Para empezar, ajustamos el modelo a la vista y seleccionamos toda la cabeza y los pinchos con la herramienta de selección rectangular *Rectangle tools*. Elegimos la textura *Skin 1* en *Texture Paint* de *Command Panel* y aplicamos la herramienta de relleno. Seguimos con el tronco. Una vez seleccionado le aplicamos un relleno con la textura *Skin 3+*, la cual hemos modificado previamente el tono ("Hue") a 61 y el tamaño horizontal ("Scale X") a 13.25. Como siempre, aplicaremos algunos detalles para darle a la planta un aspecto más real. Empezamos por perfilar la base de la planta con el pincel y eligiendo un color verdoso y el efecto de pintado *Pastel Oil* + en *Variations*. Luego pintaremos manchas de sangre en las espinas y cuerpo y algo de suciedad en el tronco con el pincel al 15 % de su tamaño y con la característica *Pastel Oil*. No olvidemos elegir el color rojo para la sangre y un tono marrón para las manchas de suciedad (Fig. 3).

Una vez hayamos terminado, exportamos la textura contenida en el canal de color con el nombre "textura_shaark.bmp" en el directorio "C:\juego_ZOF\texturizado\shaark".

MODELANDO LA PLANTA CARNÍVORA DREECKS CON MILKSHAPE 3D

Para modelar la planta tenemos que definir una postura: optamos por hacerla totalmente rígida, mirando hacia arriba y con los tentáculos de los ojos extendidos. La sensación de movimiento se construirá posteriormente con el esqueleto en Character FX.

Empezamos por la base de la planta, la cual parte de una esfera con 6 "Stacks" y 12 "Slices". Seleccionamos la mitad de la misma sin elegir *Ignore Backfaces* y cortamos. Seguidamente, el vértice superior de la mitad de la esfera lo desplazamos hacia abajo para crear el hueco donde irá el tallo (Fig. 4 – 1).

Para crear el tallo partimos de un cilindro de 4 divisiones de altura (stack) y 8 de circunferencia (slices) Estas divisiones o anillos nos servirán posteriormente para dar alguna forma al tallo (Fig. 4 - 2). Ensanchemos, pues, la base del tallo seleccionando (F1) los vértices más inferiores con la herramienta de escalado (F4) en la vista superior. Hacemos la misma operación con los vértices de las divisiones impares siguientes desde abajo hacia arriba. Para la última división superior realizamos un escalado mayor que en las demás hacia fuera, ya que servirá como apoyo a la siguiente pieza que es la base de la boca (Fig. 4 - 3). Para esta pieza, utilizamos una geoesfera de densidad 1 y le

borramos la mitad superior de los vértices. A continuación, ajustamos la esfera a la parte superior del tallo, la cual habíamos ensanchado. El ajuste lo realizamos mediante operaciones de escalado y desplazamiento (Fig. 5 - 1).

Para modelar las hojas vamos a utilizar tres cubos aplastados, alargados y unidos entre sí. Sólo es necesario modelar uno de ellos con la herramienta de escalado. Los demás los iremos duplicando ("Ctrl" + "D") y colocando uno a continuación del otro para poderlos unir (Fig. 5 – 2,3). La unión la realizamos seleccionando el primer vértice superior de cada cubo ignorando los de atrás (*Ignore Backfaces* deseleccionado) y pulsando "Ctrl" + "D" (Snap Together). Hacemos lo mismo con los vértices inferiores. A continuación, elegimos los vértices traseros con *Ignore BackFaces* seleccionado y los pegamos también. El resto de los cubos los unimos de la misma manera (Fig. 6 – 1).

Para dar la forma a las hojas a partir de los tres cubos unidos, utilizaremos las herramientas de escalado y desplazamiento en los vértices que hemos unido. Por ejemplo, para obtener la punta de la hoja, seleccionaremos los vértices más exteriores en la vista superior y los escalamos hacia dentro (Fig. 6 – 2). Para obtener las formas curvas, elegiremos los vértices del centro, también en la vista superior, y los desplazaremos hacia un lado. Estos procedimientos se muestran gráficamente en la figura 7.

Para la otra hoja sólo tenemos que duplicar la que tenemos, seleccionándola completamente y pulsando "Ctrl" + "D". Seguidamente, hacemos un *Mirror Front <-> Back* en el menú *Vertex* Para evitar que sea igual que la primera, la modificaremos un poco mediante el desplazamiento de sus vértices y la colocaremos un poco más abajo en el tallo.

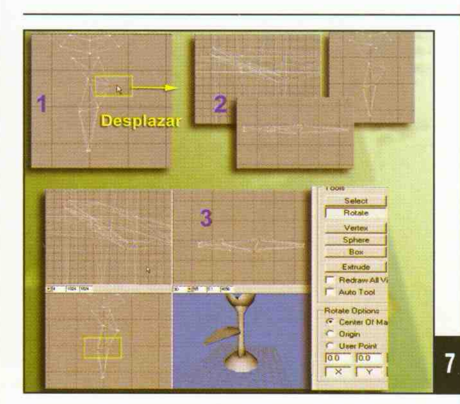
La boca del Dreeck está compuesta por dos grandes mandíbulas. Para modelarla, necesitaremos una esfera de 6 "Stacks" y 12 "Slices" dividа en dos. Para ello, tenemos que seleccionar la mitad de los vértices con *Ignore Backfaces* deseleccionado y borrarlos. Nos quedamos entonces con la mitad de la esfera que corresponderá a una de las mandíbulas. Sólo es necesario trabajar con una de ellas, ya que la otra la podemos obtener mediante un duplicado (Fig. 8 – 1,2).

Hay que tener en cuenta que, al cortar una esfera, la parte interior queda vacía; es decir, no hay ninguna cara poligonal y necesitamos cerrarla. Si cambiamos la vista de perspectiva en modo "Flat Shade" podemos observar este hecho (Fig. 8 – 3). Para cubrir el hueco dejado, necesitamos fabricar manualmente los polígonos que faltan. Partiremos, para realizar esta operación, de los vértices que forman el contorno interior de la semiesfera. Los seleccionamos y los desplazamos hacia un lado para poder unirlos. Para ello, elegimos la herramienta *Face* del grupo *Model* y unimos cada vértice entre sí de tres en tres para formar los polígonos que cerrarán la semiesfera (Fig. 9).

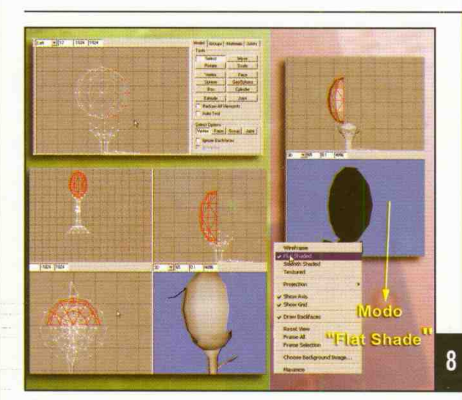
Una vez completada la operación, volvemos a colocar la cara interior de la semiesfera en su sitio seleccionando de

NOTA

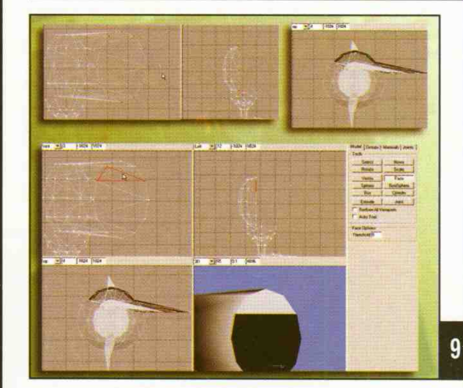
Hay una opción en Milkshape 3D que nos permite saber la cantidad de polígonos que tiene nuestro modelo. Se encuentra en el menú *Tools* y la opción es *Show model Stadistics* (023).



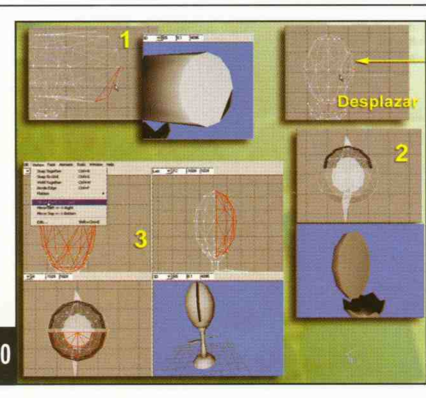
Mediante el desplazamiento individual de vértices podemos dar variadas formas a las primitivas.



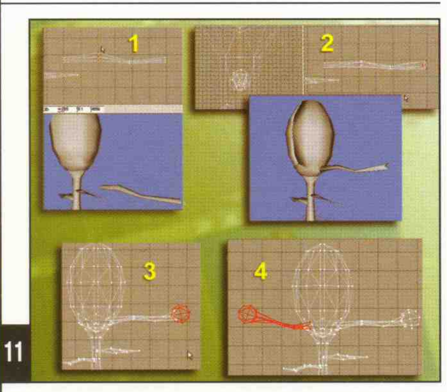
La mitad de una esfera puede servir perfectamente como mandíbula.



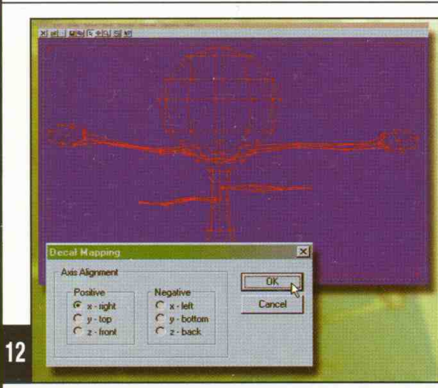
Una vez cortada la esfera es necesario cubrir de polígonos el hueco interior para obtener adecuadamente una de las mandíbulas.



Una vez cerrada la semiesfera, la duplicamos para crear la otra mandíbula.



A partir de un cilindro y por medio de los procedimientos utilizados para crear las hojas es posible obtener los tentáculos oculares.



Antes de texturizar la planta es necesario crear el mapeado UV, en este caso con LithUnwrap.

nuevo todos los vértices y desplazándolos a su lugar original (Fig. 10).

El siguiente paso será ajustar la mandíbula a la base de la boca con las herramientas de escalado y desplazamiento. Una vez ajustada, duplicamos y hacemos un espejo horizontal (*Mirror Front<->Back*) para generar y colocar la otra mandíbula (Fig. 10 –3).

Las últimas partes que nos quedan son los tentáculos que albergarán a los globos oculares. Partimos de un cilindro con 3 partes que colocaremos horizontalmente en la vista "Left" rotándolo 90 grados. Para dar forma al cilindro y obtener un aspecto más orgánico, seleccionamos los vértices de cada parte del cilindro y los desplazamos al igual que hicimos con el modelado de las hojas. Debemos también preparar el extremo del tentáculo para albergar el globo ocular. Así que, mediante el desplazamiento de los tres vértices abrimos el extremo del cilindro como se muestra en la figura 11.

Para el otro tentáculo, realizamos de nuevo un duplicado y una simetría horizontal y ajustamos la nueva posición.

Para los globos oculares utilizamos una esfera de 4 "Stacks" y 4 "Slices". Para terminar: le asignamos un tamaño adecuado, la colocamos en el extremo del tentáculo, duplicamos y aplicamos de nuevo una simetría horizontal.



RECORDATORIO

Si al unir tres vértices para formar un polígono éste no se ve en la vista de perspectiva es que el orden de dichos vértices está al revés. Para solucionarlo, basta con elegir la opción "Face / Reverse Vertex Order".

Finalmente, debemos agrupar todas las partes, seleccionando el modelo completo y pulsando el botón "Regroup" en la herramienta de agrupamiento *Groups* (Fig.11 - 3,4).

Ya sólo nos queda salvar el modelo terminado en formato .OBJ para poder texturizarlo con Deep Paint 3D.

CREANDO EL MAPEADO UV

Antes de crear la textura de la planta es preciso general su mapeado UV. Aunque es posible hacerlo con Deep Paint 3D a través de la herramienta "MercatorUV", utilizaremos LithUnwrap, ya que nos permitirá también aplicar un optimizado de vértices.

Como es habitual, dentro de LithUnwrap, cargamos el modelo y elegimos la opción *Optimize Model* en el menú *Tools*. Una vez realizada la optimización, seleccionamos el objeto por completo con *Select / All* y de nuevo en el menú *Tools* elegimos *UV Mapping / Decal* y pinchamos la opción *Positive / x- right* para alinear los ejes hacia un lado.

Finalmente, exportamos el modelo en formato .OBJ y ya estamos preparados para texturizar nuestra planta carnívora (Fig. 12).

RECORDATORIO

UV Mapping o "mapeado de coordenadas UV" es una serie de posiciones colocadas en una imagen que se unen a puntos en un objeto 3D para ubicar una textura sobre él.



En el próximo número...

... realizaremos el texturizado y la animación del Dreeck.

Herramientas para hacer la música de un **juego** (II)

En el número anterior comenzamos a desarrollar un loop en Fruity Loops. Estudiamos lo fácil que es crear y copiar patrones, los cuales utilizaremos para componer la canción que servirá de loop. En esta entrega terminaremos nuestro loop y seguiremos conociendo nuevas características que hacen de esta herramienta una de las más sofisticadas e interesantes.

UTILIZANDO "PLAYLIST"

En Fruity Loops, la canción se escribe en la sección "Playlist". En ella, se crea una serie de listas con el orden de reproducción de cada patrón a lo largo de una línea de tiempo. Para visualizar "PlayList" elegimos la opción "View / Playlist" o bien "Ctrl" + "L". Es preciso activar el modo "Loop" para poder trabajar en "Playlist". Una vez activado, la reproducción ("Play") sólo afectará a "Playlist" y no a las pistas de patrones. "Playlist" está dividido por una cuadrícula. Cada cuadro corresponde a un compás. Horizontalmente lo forman los diferentes compases a lo largo del tiempo y verticalmente los diferentes patrones. De este modo, para construir la canción sólo tenemos que rellenar estos cuadros dibujando con el ratón. Podemos ver en la figura 1 la composición en "Playlist" para nuestro loop (este loop esta contenido en el CD con el nombre "loop_fruity.wav") (Fig. 1).

Una vez terminado el loop, sólo queda exportarlo en formato .WAV (aunque también es posible hacerlo en .MP3) Elegimos la opción *File / Export / Wave File ...* o pulsamos "Ctrl" + "R". Aparecerá una ventana flotante, donde podremos elegir la calidad para nuestro loop.

EDITANDO LAS PISTAS

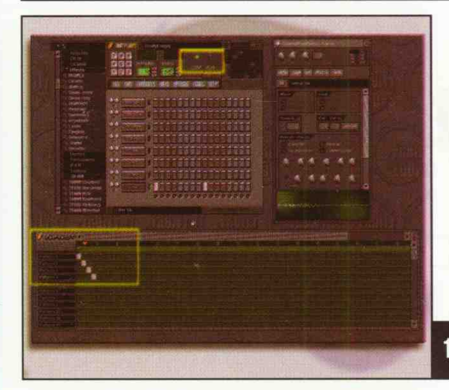
La posibilidad que Fruity Loop admite para la edición de las pistas es sorprendente y con un diseño eficaz. Antes de continuar, apuntar que en Fruity Loops las pistas de sonido se denominan *canales*. Disponemos de un editor gráfico que nos permite modificar los parámetros del volumen, panorámico, tono, frecuencia de corte, etc.. de cada canal. Con sólo pulsar sobre el botón aparecerá debajo del canal seleccionado un panel donde, mediante un lápiz, podemos dibujar los nuevos parámetros pulsando el botón izquierdo del ratón. Para borrarlos, utilizaremos el botón derecho. En la parte inferior del panel se encuentra un deslizador, que sirve para elegir los diferentes parámetros.

Para seleccionar otro canal, basta con pulsar en el "led" o lucecita verde que hay en el comienzo de los compases y para mutarla o desactivarla en la luz que se encuentra debajo de los botones de volumen y panorámico.

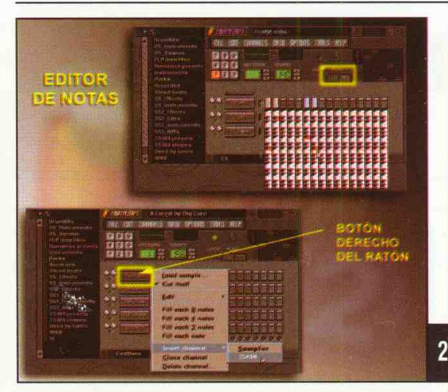
También disponemos de un editor gráfico de notas, el cual aparecerá debajo del canal seleccionado al pulsar el botón . Este panel tiene forma de pequeños teclados en el que podemos seleccionar el tono de nuestra nota (Ver Fig. 2).

TIPOS DE CANALES

Fruity Loops trabaja con dos tipos diferentes de canales según su contenido: canales de muestras ("sampler") y canales de sonidos sintetizados ("TS404") El



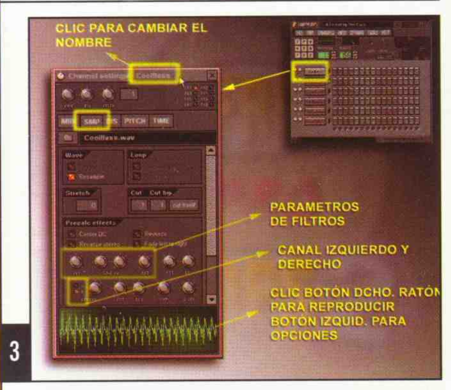
1 *En la sección "Playlist" es donde escribimos la canción que servirá de loop.*



2 *Editor gráfico de notas y procedimiento para insertar un nuevo canal.*

NOTA

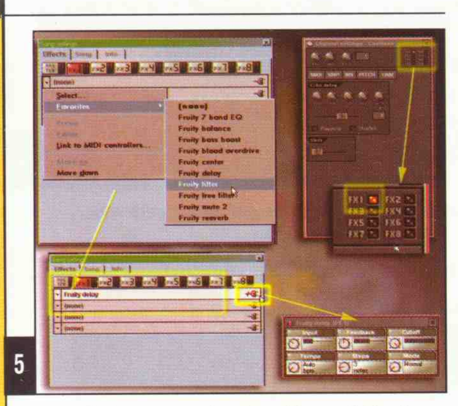
Es posible cargar varias muestras a la vez, seleccionándolas en el directorio con la tecla "Shift" pulsada. Al cargarse, los ficheros cargados se repartirán en canales consecutivos.



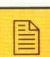
Descripción de los principales parámetros del editor de muestras.



Descripción de los principales parámetros del sintetizador TS404.



Procedimientos para la asignación y edición de efectos.



TRUCO

Para modificar los parámetros de todas las notas a la vez de una pista, mantenemos pulsada la tecla "Ctrl" mientras dibujamos con el lápiz.

comportamiento es el mismo para los dos, la diferencia estriba en el tipo de sonido que reproducen. Los canales de muestras pueden reproducir ficheros de sonidos en formato .WAV, .XI (Fast Tracker), DS, (DrumSynth) y .SYN (SimSynth) y los canales TS404 reproducen sonidos creados con el sintetizador de bajos TS404.

Para insertar un nuevo canal elegimos la opción *Channel / Add one* y el tipo de canal ("Sampler"o "TS404"). También podemos hacerlo en el menú flotante que aparece al pulsar con el botón derecho del ratón sobre el botón que muestra el sonido en cada pista. En el mismo menú, también es posible realizar operaciones de clonado y borrado de canales, así como funciones de edición básicas.

MODIFICANDO SONIDOS

Al pulsar sobre el botón se abrirá una nueva ventana llamada "Channel settings" en donde podemos modificar todos los parámetros del sonido.

EDITANDO MUESTRAS

En la ventana "Channel settings", la pestaña "SMP" corresponde a los parámetros de una muestra. Desde ahí podemos cargarla, crear una nueva, modificarla o grabar la nueva muestra modificada. En la figura 3 se describe este panel.

Pulsando sobre la gráfica de la muestra la reproducimos y haciendo clic con el botón derecho del ratón, mostramos las opciones para cargarla, crearla o grabarla.

Para modificar su sonoridad tenemos varios parámetros. En primer lugar, disponemos de tres filtros ("FX1", "Sine FX" y "FX 3"), los cuales se utilizan para cambios de volumen y modulación de la onda. Además, disponemos de un amplificador ("Amp"), reverberación ("Reverb"), frecuencia de corte ("Cut"). También es posible modificar el ataque ("Att"); es decir, acentuación del comienzo y el decaimiento o fade ("Dec"). En "Reverse" cambiamos el orden de la muestra y las luces rojas "A" y "B" conmutan entre el canal izquierdo y el derecho del estéreo.

EDITANDO SONIDOS DE LA TS404

Si el canal seleccionado contiene un sonido TS404 la ventana *Channel settings* adquiere otro aspecto. En ella, se muestran todos los parámetros propios de un sintetizador (pestaña TS404).

Básicamente se trata de un sistema de síntesis para líneas de bajos. Consta de dos osciladores (OSC1 y OSC2), que son mezclados en el apartado "OSC 1+2". También podemos modificar la envolvente del sonido mediante los parámetros "Att" (Attack), "Dec" (Decay), "Sus" (Sustain) y "Rel" (Release).

Disponemos, además, de 4 filtros (LP12, LP24, BP y HP) en el apartado "Filter" y de un oscilador (LFO) para hacer oscilar el tamaño del primer oscilador o los filtros. Y por último, podemos aplicar a toda la onda distorsión (Dist) y eco (Delay).

A través de la pestaña "PITCH" podemos modificar el tono de la onda con valores absolutos ("Middle note") o relativos ("Fine") (Ver Fig. 4).

AÑADIENDO Y MODIFICANDO EFECTOS A LAS PISTAS

Para ambos tipos de canales es posible la aplicación de múltiples efectos. Éstos se eligen y asignan en la ventana *Song settings*, la cual se abre al pulsar con el botón derecho del ratón sobre las luces "FX1" hasta "FX8" de la ventana *Channel settings*. Es posible mezclar hasta 4 efectos a la vez para un canal de efectos. Los elegimos pulsando sobre la pequeña flecha invertida junto a "(none)" en "Select" o "Favorites" y lo activamos o desactivamos pulsando en el pequeño "enchufe" de la derecha. Una vez elegido el efecto, aparecerá una nueva ventana que muestra todos los parámetros de dicho efecto (Ver Fig. 5).

Funciones 3D (II). Texturizado de objetos 3D y brushes

La creación de un modelo 3D empieza con su modelado y termina definiendo su aspecto final mediante la aplicación de texturas.

Como recordaremos, una textura es una imagen 2D, diseñada con una aplicación de dibujo que cubre un objeto 3D, dándole a éste su apariencia definitiva. Blitz3D permite crear texturas, cargarlas de disco o modificarlas. En este número aprenderemos todo lo necesario para texturizar nuestros modelos y pintarlos con el uso de *brushes*.

CREANDO TEXTURAS

Se pueden crear texturas manualmente desde programación o bien a través de una imagen.

FICHEROS DE IMÁGENES COMO TEXTURAS

Este primer método es el más sencillo y consiste en crear la textura a partir de un fichero de imagen (.BMP, .PNG, .JPG, .PCX, o .TGA) Se utiliza la función:

```
LoadTexture
(Nombre_fichero_imagen
[,flags])
```

El parámetro "flags" o banderas permite la aplicación de efectos a la textura una vez cargada. Estos efectos se pueden combinar sumando banderas. En la figura 1 se muestra un esquema con las diferentes banderas y su efecto asociado. En el ejemplo:

```
Textura= LoadTexture
("C:\mi_textura.png",3);(1+2)
```

La variable "Textura" manipulará la textura creada a partir de la imagen "mi_textura" que se ha cargado con color (bandera 1) y canal alfa (bandera 2) (1+2=3).

TEXTURAS ANIMADAS A PARTIR DE IMÁGENES

Un claro ejemplo de cómo B3D transforma las imágenes contenidas en ficheros a texturas lo encontramos en la función "LoadAnimTexture". Este comando permite la creación de texturas animadas por medio del encadenamiento de imágenes (frames) contenidas en un fichero (Ver Fig. 2 y "ejemplo1.bb").

Para crear este tipo de textura es necesario tener en un mismo fichero, y de forma consecutiva, las diferentes imágenes que forman la secuencia de la animación. Una vez cargada la imagen con "LoadAnimTexture", B3D separa la secuencia en fotogramas independientes y asigna la animación como textura.

1: COLOR (por defecto)
2: CANAL ALFA (Colores alfa serán transparentes)
4: MÁSCARA (Las áreas con color no se dibujarán)
8: MIPMAPPED (Textura con bajo detalle y suavizado a grandes distancia)
16: UNIR COORD. TEXT. HORIZONTAL (U) (Evita el enrrolamiento de la textura)
32: UNIR COORD. TEXT. VERTICAL (V) (Evita el enrrolamiento de la textura)
64: MAPEADO DE REFLEXIÓN ESFÉRICA (Mapeado de entorno)

1

Lista de los parámetros "flags" posibles para la carga de texturas.



TRUCO

El mejor formato para utilizar texturas en Blitz3D es .PNG. Ocupa poco espacio en disco al igual que .JPG, comprime con calidad parecida a .BMP y admite canal alfa (transparencia).



NOTA

El formato .JPG no soporta canal alfa, mientras que .PNG y .TGA sí. Si utilizamos un formato sin canal alfa, Blitz3D lo emulará creando uno propio por medio de valores de color.



NOTA

El consumo de memoria al cargar texturas no depende del formato utilizado. Si cargamos texturas en .JPG no significa que ocuparán menos memoria por estar comprimidas. Ocuparán lo mismo que si están en formato .BMP. La explicación de esto es que Blitz3D utiliza su propio formato; es decir, cuando se carga una imagen para convertirla en textura, independientemente del formato, Blitz3D convierte esa imagen en una textura con su propio formato interno.
La única diferencia que encontramos en el uso de diferentes formatos está en el tiempo de carga. Por ejemplo, una textura en formato .BMP tardará más tiempo en cargarse que la misma en .JPG ya que, al no estar comprimida, el fichero es mayor.



Esquema de cómo construir una textura animada con un programa de dibujo.

```
LoadAnimTexture (Textura$,
Flags, Tamaño horizontal
fotograma, Tamaño vertical
fotograma,
    Primer fotograma, Número
    de fotogramas totales)
```

Al igual que en la función "LoadTExture", es posible aplicar los mismo efectos por medio de "Flags". Es importante definir el tamaño que tendrá cada fotograma exactamente igual a como lo hayamos dibujado; es decir, si la secuencia de imágenes la forman cuadros de 64 x 64 debemos colocar en "Tamaño horizontal" y "Tamaño vertical" un 64. El "primer fotograma" corresponderá al 0. Si en el "Número de fotogramas totales" ponemos un valor inferior al número de imágenes contenidas en la secuencia del fichero, B3D rellenará los huecos con blanco. Por ejemplo, disponemos de un fichero en donde hemos dibujado una secuencia de imágenes para la textura del agua. Este fichero contiene 10 fotogramas (0-9) de tamaño 64 x 64:



```
Textura_animada=
LoadAnimTexture
("Secuencia.png",
3,64,64,0,9)
```

Es importante tener en cuenta que en el ejemplo anterior la imagen del fichero original debe tener un tamaño total de 640 X 64 (10 Cuadros de 64 = 640 x 64 de altura). Si no fuera así, ocurriría un error.

CREANDO TEXTURAS MANUALMENTE DESDE PROGRAMACIÓN

Otra posibilidad muy flexible e interesante que permite B3D es la creación de texturas a partir de cero y dibujarlas a través del buffer de texturas "TextureBuffer".

```
TextureBuffer (Textura
[,fotograma])
```

Para crear una textura nueva utilizamos la función "CreateTexture":

```
Nueva_Textura=CreateTextura
(Tamaño horizontal en píxel,
Tamaño Vertical [,flags]
[,fotogramas])
```

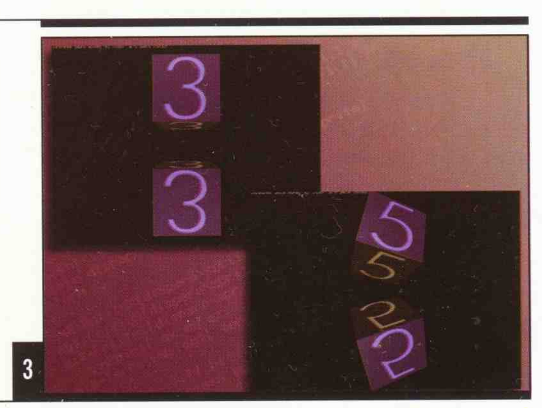
Las opciones de efectos aplicados según "flags" son las mismas que en las funciones anteriores (ver Fig. 1) y la opción "Fotogramas" indica el número de fotogramas que tendrá la textura si así fuese. Por defecto su valor es 1.

Podemos crear el tamaño de texturas que queramos. El único límite lo impone la tarjeta gráfica. De todas formas, si la tarjeta no soportara el tamaño que hayamos especificado, B3D asignará automáticamente el más cercano disponible. También, normalmente se usan texturas cuadradas con tamaños múltiplos de dos iguales; es decir, 256 x256, 64 x64. Aunque es lo más normal –ya que lo soportan todas las tarjetas-, es posible asignar tamaños diferentes como 256 x 128 o 512 x 64, siempre y cuando los valores sean múltiplos de 2.

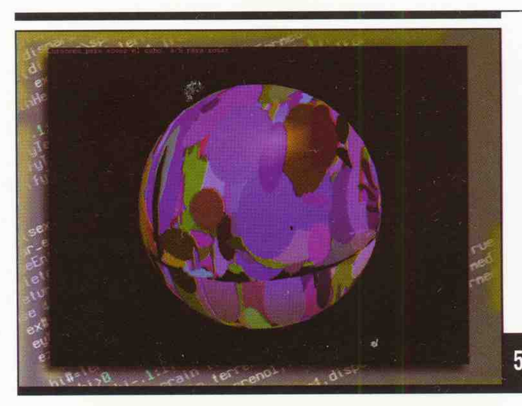
Para dibujar graficos 2D en la textura primero debemos activar el búfer de textura: *SetBuffer TextureBuffer* Seguidamente, dibujaremos la imagen y, por último, la copiamos al búfer de texturas. Se puede ver el funcionamiento de este proceso en los ejemplos "ejemplo2.bb", "ejemplo3.bb" y "ejemplo4.bb".



Textura aplicada a un cubo, la cual ha sido previamente dibujada desde programación en el búfer de texturas.



Captura sobre un ejemplo de aplicación de texturas animadas.



Ejemplo del proceso de dibujar en el búfer de textura en tiempo real.

6

Es posible escalar las texturas en cualquier dirección.

Si lo que deseamos es dibujar gráficos 3D, no podemos hacerlo directamente en el búfer de texturas. Tenemos primero que dibujar en el "BackBuffer" y luego con la instrucción "CopyRect" copiarlos al "TextureBuffer".

MANIPULANDO TEXTURAS

Disponemos de multitud de comandos para el manejo de texturas, desde escalarlas hasta cambiar sus coordenadas en el modelo.

Para cambiar el tamaño de una textura utilizaremos la instrucción "ScaleTexture":

```
ScaleTexture Textura,
Escalado U (Horizontal),
Escalado V (Vertical)
```

(Ver "ejemplo5.bb").

Para rotar una textura aplicada a un modelo utilizaremos "RotateTexture":

```
RotateTexture Textura, Ángulo
```

(Ver "ejemplo6.bb").

Esta instrucción es de



NOTA

Una vez aplicada la textura al modelo podemos borrarla de la memoria con el comando "FreeTexture Textura". Una vez borrada no podrá ser usada de nuevo; sin embargo, el modelo seguirá texturizado.

acción inmediata y resulta muy útil para aplicar efectos a los modelos. Suele ser muy utilizada en sistemas de partículas.

SCROLL DE TEXTURAS

Hay un último comando que nos permite realizar desplazamientos de la textura aplicada a un objeto 3D, muy útil para realizar scroll de texturas y simular movimiento (Ver "ejemplo7.bb"). Por ejemplo, está muy extendido su uso en texturas utilizadas para simular la superficie del agua. En el "ejemplo8.bb" se muestra la aplicación de este sistema para simular un río de lava.

MEZCLANDO TEXTURAS; "BLENDING"

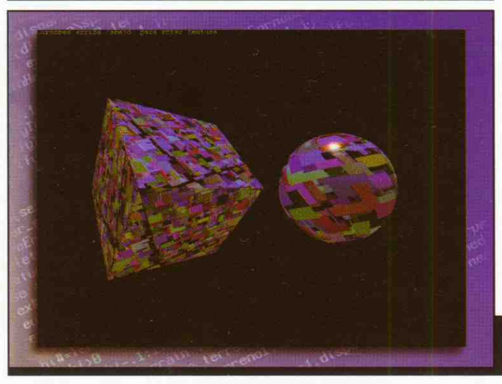
La técnica "blending" establece la forma en que la textura se mezclará con el objeto u otra textura (en caso de multitexturizado). Es posible mezclar hasta 8 texturas según la tarjeta gráfica. La textura número 0 se mezclará con los polígonos del objeto aplicado, la textura 1 con la textura 0 y así sucesivamente.

En B3D es posible aplicar esta técnica mediante la instrucción "TextureBlend": (Ver "ejemplo9.bb")

```
TextureBlend Textura,
Modo de mezcla
```

El parámetro "Modo de mezcla" determina el modo "blending" que B3D aplicará:

- **Valor 0:** no se aplicará ninguna textura.
- **Valor 1:** no se realiza mezcla alguna o sólo el canal alfa si lo hubiera.
- **Valor 2:** se multiplica la mezcla. Es el valor por defecto.



7

Ejemplo de rotación de texturas sobre meshes.

- **Valor 3:** se suma la mezcla; es decir, en multitexturizado, se suma una textura con otra.

CREACIÓN DE BRUSHES

Un *brush* o pincel es un grupo de propiedades (color, brillo, textura, etc.) que pueden ser aplicadas a una entidad, superficie o *mesh*. Los *brushes* se utilizan, por ejemplo, para cambiar el aspecto de algunos polígonos del modelo, ya que, con funciones de entidad como "EntityColor", cambiaremos el color el objeto completo.

Para asignar o pintar con un *brush* a un *mesh* utilizamos *PaintMesh Mesh, Brush*.

Para hacer lo propio con una entidad: *PaintEntity Entidad, Brush* y para pintar una superficie: *PaintSurface Surface, Brush*.



8

Un ejemplo práctico sobre la técnica de scroll de texturas para simular el fluir de un río de lava.

Ejemplos de la técnica "blending" (mezcla) aplicada a texturas.

CREANDO *BRUSHES* MANUALMENTE

Para crear un *brush* desde cero utilizaremos la instrucción "CreateBrush":

```
Nuevo_Brush= CreateBrush
( [componente color rojo#]
[, verde#][,azul# ] )
```

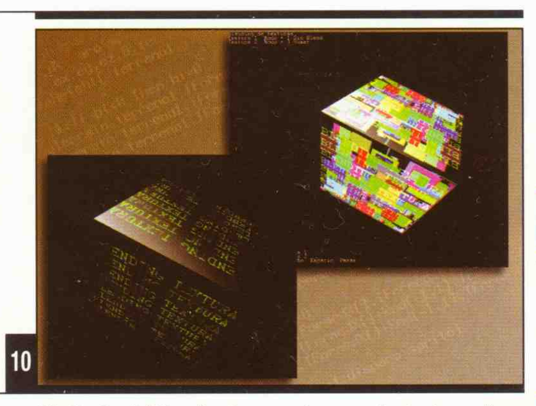
Podemos crear un *brush* nuevo y posteriormente asignarle una textura con el comando "BrushTexture":

```
BrushTexture Brush, Textura
[,fotograma de la textura]
[, índice de la textura]
```

Ejemplo:

```
Textura=LoadTexture("mi_foto.png")
Nuevo_Brush=CreateBrush()
BrushTexture Nuevo_Brush, Textura
```

Es posible asignar hasta 4 capas (0-3) de texturas por cada *brush* por lo que se pueden obtener muchos más efectos.



Mediante el "blending" se pueden mezclar texturas de diversas maneras.

CREANDO *BRUSHES* A PARTIR DE FICHEROS DE IMÁGENES

Con la instrucción "LoadBrush" podemos cargar un fichero de imagen, crear una textura con él y asignarla posteriormente a un *brush*:

```
Nuevo_Brush= LoadBrush
(Imagen[,Flags][,Escala U]
[,Escala V]
```

Las opciones "Flags" son las mismas que las de las texturas (ver Fig. 1).

Además, podemos definir directamente el escalado que tendrá la textura aplicada al *brush*.

APLICANDO EFECTOS A LOS *BRUSHES*

Al igual que ocurre con las texturas, es posible aplicar ciertos efectos a los *brushes* como brillo, color, etc.

Para asignar un color a un *brush* utilizaremos:

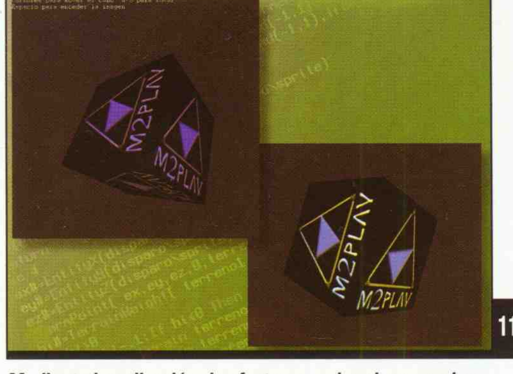
```
BrushColor Brush, Componente
color rojo#, verde#, azul#
```

Se puede obtener un brillo más intenso en ciertas partes del modelo donde la luz incide por medio del comando:

```
BrushShininess Brush,
Cantidad de brillo# (0-1)
```

También es posible utilizar "blending" en los *brushes* con:

```
BrushBlend Brush,
Modo de mezcla.
```



Mediante la aplicación de efectos a un brush se pueden obtener resultados muy interesantes.

Los modos de mezcla son: el 1 canal alfa, 2 multiplicar y 3 sumar.

Hay una instrucción muy interesante que es "BrushAlpha", y se puede utilizar para ocultar una entidad en el lugar de "HideEntity", en el caso de que se quiera seguir detectando por el sistema de colisiones.

Por último, disponemos de otro comando para la aplicación de efectos extras:

```
BrushFX Brush, Efecto
```

Los parámetros de efectos pueden ser sumados para combinar efectos.

Los valores posibles para "Efecto" son:

- **0:** Por defecto (ningún efecto).
- **1:** Brillo.
- **2:** Usar "Vertex Colors" en vez de "Brush Color".
- **4:** Sombreado plano.
- **8:** El efecto de niebla no afectará al modelo.
- **9:** Desactiva la ocultación de los polígonos traseros. Se muestran todos los polígonos en modo "Wireframe" (alámbrico) (Ver "ejemplo10.bb").

En el próximo número...

... estudiaremos el sistema de animación y las técnicas más recomendadas.

Cómo manejar niveles BSP

Empezamos una serie de tutoriales relacionados con el desarrollo y programación de juegos. Dedicaremos este espacio para construir rutinas útiles.

Además, responderemos a cuestiones que quedaron pendientes durante el curso y ampliaremos otras que, por su interés, merecen un estudio más a fondo. Para empezar, estudiaremos una característica muy interesante contenida en Blitz3D: el manejo de niveles BSP.

MUNDOS BSP

La nueva posibilidad que Blitz3D, a partir de su versión V. 1.76, brinda para el uso de niveles BSP, posibilita la creación de una manera sencilla de juegos al estilo Quake. Antes de seguir, recordemos qué significa BSP y en qué consiste esta técnica:

BSP son las siglas de "*Binary Space Partitioning Tree*" – árbol binario para la partición de espacio- y es, básicamente, una forma de organizar objetos dentro de un espacio. Esta técnica aplicada a un entorno 3D coloca los objetos en una estructura de árbol que es seguida por el programa de forma ordenada y decide qué polígono se debe mostrar y cuál no. Es un buen método para eliminar del campo de visión áreas ocultas, con lo que se aumenta en gran medida el rendimiento total.

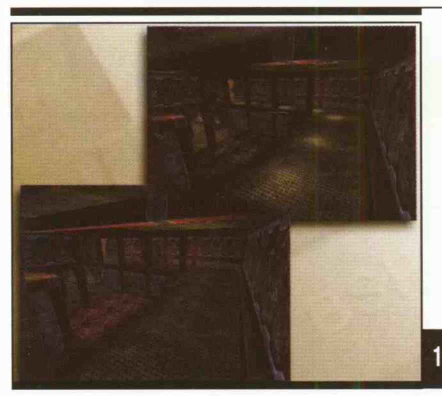
Esencialmente, todo el trabajo que representa el manejo de árboles BSP lo resuelve Blitz3D para el usuario mediante su filosofía de entidades. Esto quiere decir que se puede cargar un nivel BSP completo y asignarlo a una variable como si de un modelo 3D se tratara. Así que, al ser una entidad, se le pueden aplicar los comandos básicos de éstas como posicionamiento, escalado o rotación, así como todas las funciones completas de colisión.

```
LoadBSP ( "Nivel.bsp"
[, ajuste gamma] [,madre])
```

La opción "ajuste gamma" nos permite modificar la intensidad de luz del mapeado de luces del nivel BSP. El valor por defecto es 0 y va desde 0 hasta 1.

Ahora bien, hay ciertos puntos que debemos tener presentes a la hora de utilizar modelos BSP:

- **1.** Los modelos BSP no pueden ser texturizados, pintados, coloreados, etc... desde programación.
- **2.** Las texturas utilizadas en un nivel BSP deben ir en el mismo directorio que el modelo.
- **3.** Los modelos BSP no son iluminados por las luces ambientales del Blitz3D. Así que debemos utilizar la iluminación propia para el modelo BSP ("BSPAmbientLight").
- **4.** Si aplicamos "AmbientLight" normal del Blitz3D, éste se sumará al "AmbientLight" del nivel BSP, de tal manera que podemos jugar con estos dos valores para obtener algún tipo de efecto de iluminación.

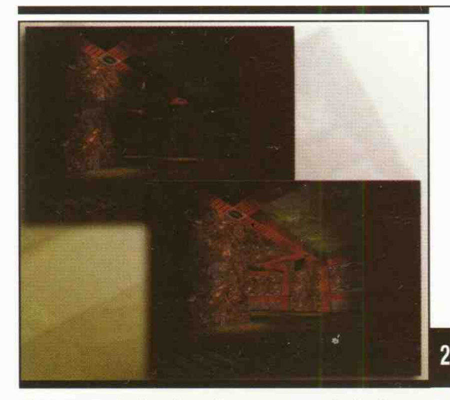


Se muestra la activación de "LightMapping". Y en la imagen inferior "Vertex Lighting".

```
BSPAmbienLight variable que
contiene el modelo BSP, Valor
de Rojo#, Valor de Verde#, Azul#
```

- **5.** Es posible definir qué sistema de iluminación afectará al mundo BSP entre mapeado de luces (LightMapping) o iluminación de vértices (Vertex Lighting). Este último suele ser más rápido en algunas tarjetas gráficas pero no ofrece la misma calidad que un mapeado de luces.

```
BSPLighting variable que
contiene el modelo BSP, tipo
de iluminación (True/False)
```



Una zona sin iluminar (imagen superior) y la misma iluminada con el uso de la linterna (imagen inferior). Sistema "LightMapping" activado.



NOTA

Las cualidades del manejo de niveles BSP se encuentran disponibles a partir de la versión 1.76 de Blitz3D.



Una zona sin iluminar (superior) y la misma iluminada con el uso de la linterna (inferior). Sistema "Vertex Lighting" activado.

Si en el parámetro "Tipo de iluminación" colocamos "True" se utilizará el mapeado de luces; si por el contrario utilizamos "False" elegiremos la iluminación de vértices (Fig. 1).

Ejemplo:

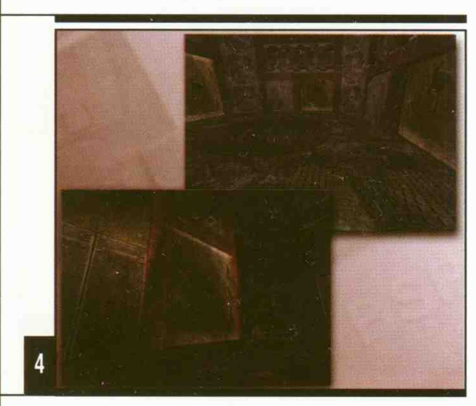
```
...
Nivel = LoadBSP("nivel1.bsp",0.5)
BSPAmbientLight Nivel,200,150,150
BSPLighting Nivel, True
...
```

UN EJEMPLO PRÁCTICO

INTRODUCCIÓN

Incluimos en el CD un ejemplo de cómo manejar niveles BSP llamado "BSP.bb y BSP.exe". En él utilizamos colisiones, iluminación y cómo implementar un sistema de puertas automáticas. A continuación, haremos una descripción del programa para esclarecer el uso de niveles BSP.

El ejemplo adjunto admite la posibilidad de navegar por el interior de un nivel BSP mediante una cámara y el uso del ratón y los cursores. También es posible activar o desactivar una linterna para estudiar las posibilidades de iluminación en tiempo real, así como la elección de los dos modos de iluminación posibles: "LightMapping" y "Vertex Lighting". Además, se incluye un sistema que utiliza una estructura de datos para colocar puertas en diferentes lugares del nivel, las cuales se abren y cierran automáticamente al ser tocadas.



Imágenes de las puertas automáticas implementadas en el nivel BSP.

DESCRIPCIÓN BREVE DEL PROGRAMA

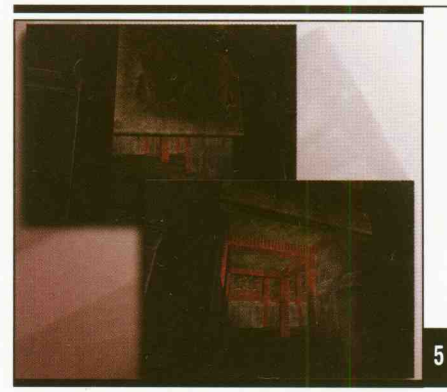
Después de definir las constantes para el sistema de colisiones y las variables necesarias, creamos la cámara que utilizaremos como entidad principal (Ver comentario"Define camara").

Para crear la linterna, utilizamos una luz de tipo "Spot" con un cono de luz de 60 grados y dependiente de la cámara: *Linterna=CreateLight(3,camara): LightConeAngles Linterna,0,60* (Ver Fig. 2 y 3).

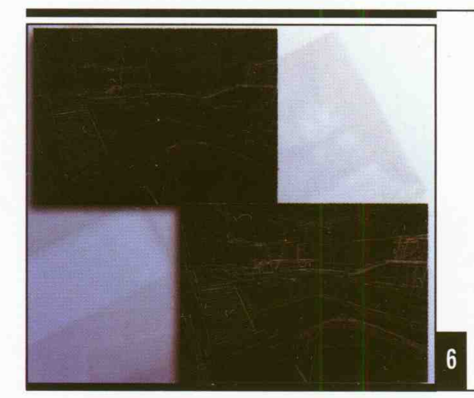
Creamos una puerta a partir de un cubo escalado. Esta entidad nos servirá luego para hacer las copias de las demás puertas. Seguidamente, definimos una matriz (contendrá las posiciones y el estado de cada puerta) y su estructura correspondiente (Ver "Define puertas"). Después de cargar y definir el nivel BSP, creamos el mapa de colisiones y llamamos a la función para crear las puertas *Crear_Puertas()*. En esta función, simplemente, añadimos una nueva puerta a la estructura y rellenamos los campos a partir de la matriz de datos (Ver "Función Crear Puertas") (Ver Fig. 4).

Dentro del bucle principal destacamos el procedimiento para detectar y guardar qué puerta hemos tocado (Ver "Detecta puertas"). A continuación, abrimos o cerramos las puertas dependiendo de la orden contenida en las variables *SW_Abrir* y *SW_Cerrar*. Apuntamos que la variable auxiliar *Aux_Puerta* contiene la entidad *puerta* que hemos tocado y que pasaremos luego a las funciones de apertura y cierre (Ver Fig. 5).

Para terminar, el resto de instrucciones se refiere a la activación de la linterna y de los diferentes modos de iluminación y de todo el control por el usuario de la cámara (Fig. 6).



Al chocar con las puertas, éstas se abren hacia arriba y al segundo se vuelven a cerrar.



Arriba, modo en alambre (wireframe) con "LightMapping" y abajo el mismo modo pero con "Vertex Lighting".

En el próximo número...

... explicaremos la manera de implementar los diferentes sistemas de creación de cielos para nuestros entornos.

Juegos de estrategia (II). Estrategia en **tiempo real**

Continuamos con los juegos de estrategia, conociendo la historia de un género que domina prácticamente la producción de juegos en la actualidad: los RTS o juegos de estrategia en tiempo real.

La producción de títulos de este género es enorme y se cuenta por cientos. Debido a esto, sólo mencionaremos los más significativos. También debemos hacer una distinción dentro de este género según la premisa básica del objetivo del juego; es decir, si tiene o no un carácter bélico.

UN PASO DE GIGANTE

Volvamos atrás unos años y conozcamos la chispa que generó la masiva incursión de las partidas en tiempo real para un tipo de juego exclusivo de unos pocos donde las técnicas de tablero aún dominaban al PC. Allá en los comienzos de los 90, a un señor llamado *Brett Sperry*, obsesionado por buscar la diversión en los juegos de estrategia por turnos de aquel momento, se le ocurrió la brillante idea de que las posibilidades del ordenador permitirían eliminar el sistema por turnos y hacer viable un género más atractivo, fácil y sobre todo más entretenido. Nació entonces *Dune II* en 1992, de la mano de su desarrolladora WestWood Studios, el primer juego de estrategia en tiempo real de la historia. A partir de ahí, WestWood creció meteóricamente y publicó su siguiente trabajo, *Command & Conquer*. Le siguió una saga completa basada en el mundo de C & C, la cual se convertiría en la más vendida de la historia de los juegos de estrategia y construyó las bases del género: *C & C: Red Alert I* y *II* (1997 y 2000), *C&C: Tiberina Sun* (2000) (Dunell ,C&C).

Al igual que ocurrió en el género por turnos, apareció paralelamente al ambiente bélico de C&C, un título ambientado en los mundos de *Warhammer*, *Warcraft* (Blizzard, 1994), en el que combaten orcos contra humanos. Le siguió *Warcraft II* (Blizzard, 1996), que ha llegado a convertirse en un clásico.

EVOLUCIÓN TÉCNICA

Empezamos como base dando un repaso a los títulos que usan una ambientación de conflictos armados para establecer el argumento del juego.

La aparición de la estrategia en tiempo real originó la base fundamental de este genero, la Inteligencia Artificial (IA). Hasta la llegada del 3D, la única evolución que sufrían estos juegos hacía referencia sólo el apartado de la IA. De una forma u otra la base impuesta por WestWood para sus juegos era copiada hasta la saciedad, y prácticamente la interfaz de usuario o la jugabilidad eran iguales en todos los títulos: *War Wind* (*SSI*, 1996), *KKND* (Melbourne House,1997), *Earth 2140* (Topware, 1997), *Dark Reign* (Activision, 1997), etc. Aunque también aparecieron juegos originales con otro tipo de ambientación como: *Settlers* II (Blue Byte, 1996) y sus pequeños colonos; la mezcla de rol y estrategia de *Lords of Magic* (Sierra, 1997); los robots de *Z* (BitMap Brothers, 1996) y *Sindícate Wars* (BullFrog, 1996) y, cómo no, *Sid Meier´s GettysBurg* (Firaxis, 1997), basado en la Guerra Civil Norteamericana, etc. Pero en general, quizás la única diferencia radicaba en los gráficos y en la ambientación. Desde *Dune II* hasta 1997 el género no cambió



Dune II y Warcraft fueron los primeros juegos de estrategia en tiempo real de la historia.



La serie Command & Conquer estableció las bases del género.



Z y Gettysbug fueron dos ejemplos de títulos que se distinguían por su original ambientación en una época de producción masiva.



Tres pesos pesados de la estrategia en tiempo real: Starcraft, Age of Empires y Total Annihilation.



Imágenes de los primeros juegos del género en 3D.



Giants y Sacrifice, dos joyas de arcade – estrategia.



Dos apuestas sobresalientes a la estrategia espacial en 3D.

mucho. Fue entonces cuando apareció *Starcraft* (Blizzard, 1997), considerado uno de los mejores juegos de estrategia de todos los tiempos. En este mismo año irrumpe también Microsoft con *Age of Empires* que ganó el podium de los juegos de estrategias históricos. Así mismo, en este año tan prolífico apareció un nuevo concepto de jugabilidad con el título *Total Annihilation* (CaveDog,1997).

EL PASO A LAS 3D

Una vez más, la evolución técnica en los compatibles llama a la puerta de las desarrolladoras de este género ofreciendo las nuevas tarjetas gráficas y señalando hacia el 3D. La perspectiva isométrica deja paso a la tercera dimensión y fue Bungie con su juego *Mit.: The Fallen Lords* los primeros en adoptarla en Noviembre de 1997. Un nuevo mundo se abre ante la posibilidad de visualizar los combates desde diferentes puntos de vista y de poder ver con más detalles todas las unidades. Esto provoca una nueva revolución comercial que hace que 7 de cada 10 juegos sean de estrategia. Obviamente, aparecieron nuevos conceptos y sobre todo mejoras gráficas que abarcaban todos los argumentos imaginables. Sin embargo, los clásicos se mantienen y aparecen nuevas partes y expansiones adaptadas al 3D como: la tercera parte de Age of Empires, *Age of Mithology* (Microsoft, 2001), *Warcraft 3* (Blizzard, 2001), *Dark Reign 2* (Activision, 2000) o *Emperor: Battle for Dune* (2001), primera incursión de WestWood en las 3D.

Sin embargo, la isometría seguía siendo la preferida de algunas desarrolladoras y sobre todo de los incondicionales del género y aparecieron títulos realmente brillantes como *Tzar* (Infinite Loop, 2000), *Seven Kingdoms* (Interactive Magic, 1998), o *Space Clash* (Dinamic Multimedia,1999).

A partir de 1999, la mayoría de los títulos se desarrollaban en 3D y la potencia de las máquinas hacían cada vez más viable el manejo de enormes cantidades de unidades y sobre todo un mayor espectáculo gráfico. Títulos como *Shogun: Total War* (Creative Assembly / Electronic Arts, 2000) pueden mostrar incluso hasta diez mil unidades a la vez en pantalla luchando entre sí o *Warrior Kings* (Sierra, 2001) en donde es posible admirar las batallas a pie de soldado.

Aparecen también títulos que mezclan la estrategia con el más puro arcade dando un nuevo concepto al género y proporcionando una jugabilidad fresca y asequible. Destacamos *Sacrifice* (InterPlay, 2000) desarrollado por los siempre originales chicos de Shiny Entertaiment o *Giants: Citizen Kabuto* (Planet Moon/ Interplay, 2000).

Entra en juego la ambientación espacial y futurista para muchos títulos, encabezado por el fantástico *Homeworld* (Sierra, 1999), primera incursión de la estrategia espacial en las tres dimensiones con unos resultados increíbles ya que es posible controlar las batallas desde cualquier ángulo posible. Un año después, Sierra publica *Ground Control*, basado en el éxito de Homeworld, pero esta vez en tierra firme. Siguiendo los pasos de Sierra, se publican nuevos títulos ambientados en el futuro y sobre todo en el espacio como son: *Far Gate* (Super X Studios, 2001), *Conquest: Frontiers Mars* (Ubi Soft, 2001), *Star Wars: Force Commander* (Lucas Arts, 2000), *Star Trek: Klingon Academy* (14 Degrees East/Interplay, 2000), etc.

Muchos son los conceptos diferentes en torno a la estrategia en tiempo real que aún nos quedan por ver. Subgéneros como los RPS (Role Player Strategy), la construcción de ciudades o la exclusiva gestión de recursos en todos los ámbitos conocidos, quedan pendientes.



En el próximo número...

... conoceremos más sobre la historia de los juegos de estrategia en tiempo real.

Cuestionario Videojuegos 11

Preguntas

1. ¿Cómo podemos dibujar con Blitz3D una textura en un objeto 3D en tiempo real?
2. ¿Qué son los *brushes* y cómo podemos crearlos en Blitz3D?
3. ¿Cómo podemos conseguir que nuestra nave espacial acelere al pulsar la tecla del espacio y desacelere al soltarla?
4. ¿Qué instrucción utilizaremos para que, por ejemplo, la cámara apunte siempre hacia un objeto?
5. ¿Cómo podemos cambiar la tonalidad de una textura en Deep Paint 3D?
6. ¿Cómo podemos darle la vuelta a los vértices de un polígono creado en Milkshape3D?
7. ¿Cuántos tipos de canales utiliza Fruity Loops y cuáles son?
8. ¿Cómo podemos añadir un efecto a una pista en Fruity Loops?
9. ¿Cómo podemos modificar la intensidad de luz de un nivel BSP en Blitz3D?
10. Define una luz en Blitz3D para utilizarla como linterna en un nivel BSP.

Respuestas al cuestionario 10

▷ **1.** En primer lugar creamos un *mesh* y luego una superficie:

```
nuevo_mesh=Createmesh()
Superficie=CreateSurface(nuevo_mesh)
```

Seguidamente sumamos vértices y los unimos:

```
AddVertex Superficie,-1,1,0
AddVertex Superficie,1,1,0
AddVertex Superficie,1,-1,0
AddVertex Superficie,-1,-1,0
AddTriangle Superficie,0,1,2
AddTriangle Superficie,0,2,3
```

Para finalizar, actualizamos las normales:

```
UpdateNormal nuevo_mesh
```

▷ **2.** Cargando el modelo con *LoadAnimMesh*, preguntar por la parte "cabeza" (suponiendo que se nombró como "cabeza" en el modelador) y texturizarla:

```
Modelo= LoadAnimMesh("modelo.3ds")
For n = 1 to CountChildren (Modelo)
    Cabeza = FindChild (Modelo, "cabeza")
Next
TextureEntity Cabeza,Textura_cabeza
```

▷ **3.** Con la función "CreateTimer" y "WaitTimer":

```
Timer = CreateTimer(60)    ; 60 FPS
...
While Not Keydown(1)
    WaitTimer (Timer)
    RenderWorld
Wend
```

▷ **4.** En 2D, los ejes X e Y corresponden a los ejes X y Z en 3D.

▷ **5.** En primer lugar seleccionamos todo con la opción *Select all* en el menú *Select* y a continuación elegimos el tipo de mapeado UV con la opción *UV Mapping* en el menú *Tools*.

▷ **6.** Pulsando con el botón derecho sobre la capa "C" (color) situada en el menú de materiales.

▷ **7.** En primer lugar, hay que crear patrones en el modo *Pattern*. Seguidamente los pasamos al modo *Song* donde asignaremos la longitud del loop y lo crearemos a partir de los patrones. Para finalizar exportamos la canción (loop) a disco.

▷ **8.** En primer lugar, debemos crear patrones a través de los distintos canales. Una vez creados, pasamos a la sección *Playlist* en donde definimos la duración de los patrones en la canción. Una vez construida la canción ya hemos creado el loop.

▷ **9.** Por medio de puntos de inflexión colocados en la pista. Estos puntos se unirán formando una gráfica, la cual cambia la envolvente en toda la pista.

▷ **10.** Por medio de la opción *Mix Down to File* en el menú *Edit* del multipista.

Contenido CD-ROM 11

▶ AUDIO

■ Intelliscore MIDI Converter 5.0
Convierte en MIDI cualquier fichero de audio en MP3 o WAV.

■ Super MP3 Recorder 3.0.1
Grabación de sonido desde micrófono para luego aplicarle efectos.

■ Crystal Audio Engine 0.1b
Procesa señales digitales en tiempo real usando este programa multicanal.

■ Quartet X2 Music Studio 2.0
Crea tus propias melodías con este potente y completo editor.

■ Sequbeat
Programa de secuenciación para percusión con calidad de estudio de grabación.

■ Hammerhead Rythm Station 1.0
Crea interesantes loops para tus composiciones.



▶ DISEÑO 2D

■ Focus Magic
Aplica fácilmente efectos a tus imágenes con los filtros contenidos en este programa.



■ Collage Maker
Crea tus propios collages con esta excelente herramienta en pocos minutos.

■ SnagIt 6
Captura fácilmente imágenes o vídeo con un solo click usando esta sencilla aplicación.

■ VaryView 1.1
Visualizador de imágenes, para que organices todos tus archivos gráficos.

■ Graphics Converter Pro 4.0
Práctica utilidad para editar rápidamente tus imágenes y cambiarles el formato.

■ PhotoMark 1.0
Protege tus imágenes para que nadie las use indebidamente y sin tu autorización.

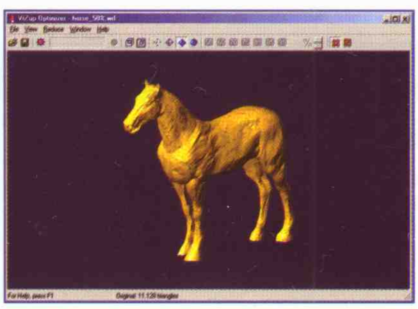
▶ DISEÑO 3D

■ 3D Exploration 1.81
Herramienta muy completa para integrar y publicar gráficos en 2D y 3D fácilmente.

■ 2D & 3D Animator 1.4
Crea, anima y edita imágenes de calidad tanto en dos como en tres dimensiones.

■ Symmetrica 3D 3.0
Poderoso programa para editar y renderizar 3D.

■ VIZup Optimizer 1.1
Herramienta de optimización 3D, para darle el máximo rendimiento a tus imágenes.



■ Meshbox 1.0
Maneja y convierte fácilmente tus contenidos en 3D.

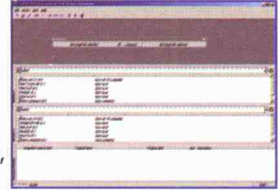
■ Vue d'Esprit 4.1
Crea, renderiza y anima escenarios naturales ultra-realistas en tres dimensiones.

▶ PROGRAMACIÓN

■ Map Master 2.0
Toma todo el control de tus mundos virtuales gracias a esta aplicación.

■ LAN-in-a-Box 1.0.3
Con esta sencilla herramienta podrás crear partidas en red para cualquier juego en equipo.

■ HexDiff 3.0
Utilísima herramienta con la que podrás controlar las versiones del código, ya que se encarga de buscar diferencias entre éstas.

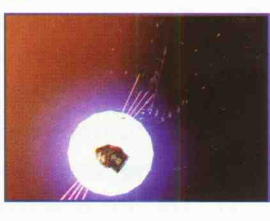


■ File Substring Replacement Utility 7.0
Otra utilidad; ésta se encarga de buscar y reemplazar strings en múltiples ficheros.

■ Elegant InterFace 4.0
Personaliza tu espacio de trabajo con elegancia e imágenes usando este programa.

▶ JUEGOS

■ Homeworld
Primera incursión de la estrategia espacial en las tres dimensiones con unos resultados increíbles.



■ Age of Empires
Estupendo juego que ganó el podium de los juegos de estrategias históricos.

■ Command & Conquer
Otro de los juegos de estrategia más vendidos de todos los tiempos.

■ Zone of Fighters
Como cada semana, nuestro juego.

▶ VÍDEO

■ DivX Pro Video 5.0.2
Con este software tendrás todo lo necesario para crear tus propios vídeos en formato DivX.



■ DVD Copy Plus
Todo lo que necesitas para copiar tus películas de DVD a CD.

■ SoftReel MPEG-2 Video Decoder 1.10
Excelente filtro de decodificación para Microsoft DirectShow (ActiveMovie).

▶ EXTRAS

En este apartado encontrarás todos los ejemplos de los que hablamos en el coleccionable, para que no pierdas detalle.